Family Name:. . . . . **Other Names:** . . . .

Student ID:. . . . . **Signature** . . . .

COMP 261 : Test 3

11 May 2023

Instructions

- Time allowed: **50 minutes**
- Attempt **all** the questions. There are 50 marks in total.
- **In-person**: Write your answers in this test paper and hand in all sheets.
  Remote: Type your answers in the template file and submit to "Test 3 Remote" on the COMP 261 submission system.
- If you think a question is unclear, ask for clarification.
- This test contributes 10% of your final grade.
- You may use dictionaries and calculators.
- You may write notes and working on this paper, but make sure your answers are clear.

Questions	Marks	
1. Network Flows	[20]	
2. Centrality	[10]	
3. Cycles and Spanning Trees	[20]	
	TOTAL:	

1. **Network Flows** **(20 marks)**

(a) Suppose that you are asked to implement the Edmond Karp algorithm to find maximum flow through a network. Using the following pseudo-code of the algorithm answer the following questions.

```
Edmond Karp(Graph, start, goal):
 // Build Residual Graph to include flow values and reverse edges
 for each edge e in G where e is of the form ⟨fromVertex,toVertex, capacity⟩
     add e ← ⟨fromVertex,toVertex, capacity,0⟩ to the Residual graph //flow value = 0 initially
     add the corresponding reverse edge e' ← ⟨toVertex,fromVertex,0,0⟩ to the Residual graph
maxflow ← 0
Repeat
     Use BFS to find a path P from start to goal in the Residual Graph such that
    all edges e in P have capacity(e)>0
     if P exists
         //Find bottleneck : the capacity of the minimum capacity edge in P
         pathFlow ← Bottleneck(P)
         Output (P, pathFlow) as an augmentation path
         maxflow ← maxflow + pathflow
         Update the Residual graph to reflect the changes in the capacities of edges
         that constitue P and the corresponding reverse edges
     else
      output Maxflow
      return
```

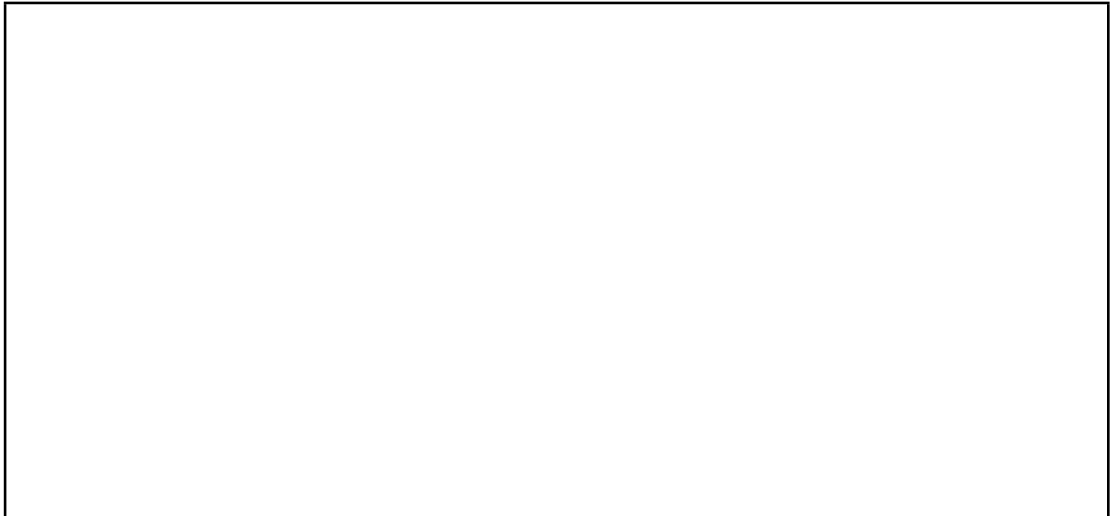
i. **(1 mark)** When running the Edmond Karp implementation based on the above pseudo-code, how does the program know when to terminate?

ii. **(5 marks)** **Adjacency matrix representation**. Consider the following adjacency matrix data structure to represent a directed graph with a capacity associated with each edge.

```
public class Graph {
   private int[ ] nodeIDs;          // Unique IDs of the nodes
   private int[ ][ ] edges;         // adjacency matrix of edge capacities
   public int nodeCount();          // returns ths number of nodes in the Graph
```

The value in edges[i][j] represents the capacity of the edge from node i to node j. The value 0 for an entry edges[i][j] indicates that either there is no edge from node i to node j or if an edge exists its capacity is 0.

Write Java Code to initialize the Residual graph for a given graph G.



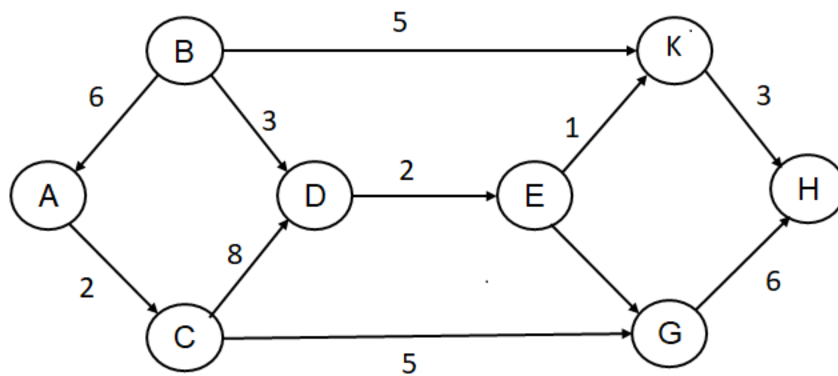
iii. **(8 marks)** Suppose you were using the above Edmond Karp algorithm to find augmentation paths from node A to node H in the graph below and the first augmentation path found by the BFS is A→ C→ G→ H.
Trace out what the algorithm will do at each of the following iterations in terms of:

- which node it will visit
- what will be added to the queue
- what items will be added to the Backpointers

(You can refer to the pseudo-code at page 10)

Assume that if at a given iteration more than one node is to be added to the queue, then they are added in alphabetical order.

The first iteration has been done for you.



Iteration 1:

Node Visited: A

Queue: C

Backpointers: add ⟨C → Edge A-to-C⟩

Iteration 2:

Node Visited

Queue:

Backpointers:

Iteration 3:

Node Visited:

Queue:

Backpointers:

Iteration 4:

Node Visited:

Queue:

Backpointers:

Iteration 5:

Node Visited:

Queue:

Backpointers:

Iteration 6:

Node

Queue:

Backpointers:

iv. **(6 marks) Adjacency list representation**.Consider the following data structures to represent a simple directed graph with capacities and flows associated with edges.

```
public class Graph{
    private Map<Integer, Node> nodes;
    private Collection<Edge> edges;
```

Edge objects have three methods:

```
    public int fromVertex();        // returns the node at the start of the edge
    public int toVertex();          // returns the node at the end of the edge
```

```
public int capacity();        // returns the capacity of the edge
public int flow();            // returns the flow of the edge
```

If the BFS on the residual graph returns augmentation path as a list of edges:

```
ArrayList<Edge> augmentationPath;
```

Write Java code to determine the bottleneck of a given path.

2. **Centrality Measures** **(10 marks)**

(a) **(6 marks)** For each of the following problem scenarios, identify the centrality measure that you want to analyse to help you make an informed decision. Give a brief explanation of your answer.

- An important information needs to be sent across a communication network and you want to send this information through the shortest paths as quickly as possible.

- You are analysing a network of computers and you want to identify nodes which would disrupt communication if they failed.

- You are analysing a social network and are required to find who is the most popular person in the network.

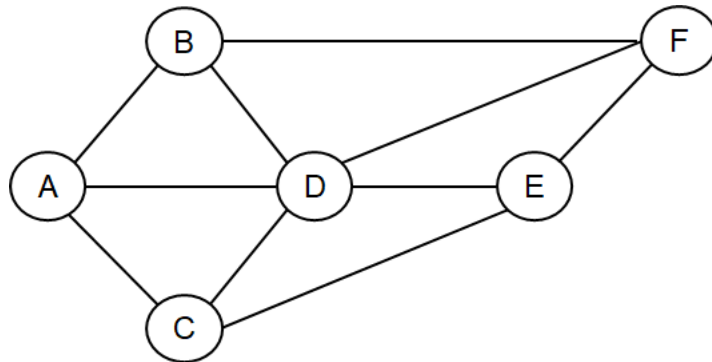
(b) **(4 marks)** Let A be a webpage on the web. Which of the following factors have an impact on the pageRank of A. Justify your reasoning in a few words.

i. Number of inbound links to A
ii. Number of outbound links from A
iii. PageRank of the webpages that are out-neighbours of A
iv. PageRank of the webpages that are in-neighbours of A
v. Number of out-neighbours of the in-neighbours of A

Note:
in-neighbours(A): Webpages that point to A
out-neighbours(A): Webpages that are pointed-to by A

3. **Cycle Detection and Spanning Trees** **(20 marks)**

(a) **(4 marks)** Consider the following graph.



Which of the following is **not** a possible order of visiting the nodes in the graph through DFS traversal (You can refer to the pseudo-code at page 10).

i. A B F D C E
ii. A B D C E F
iii. A B C D E F
iv. A B F E C D

(b) **(4 marks)** State True or False. If T is tree obtained by DFS traversal in a connected undirected graph G, which of the following statements are true.

i. A back edge is an edge that appears in G but not in T.

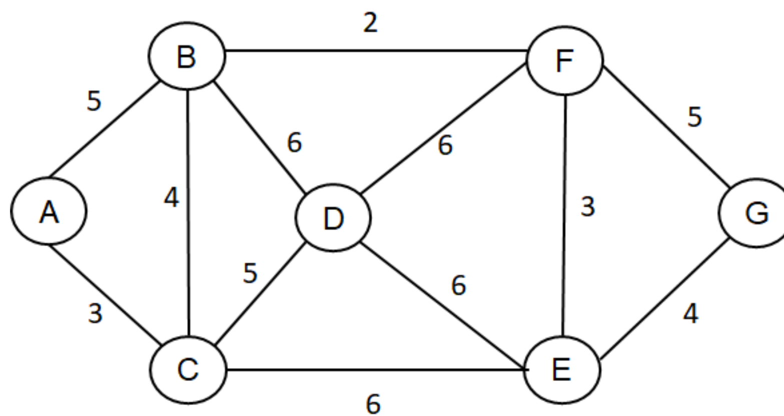
ii. T will have **atleast** n-1 edges

(c) **(4 marks)** State True or False. If S is a spanning tree of a connected undirected graph G, which of the following statements are true.

i. Removing an edge from S will disconnect the tree.

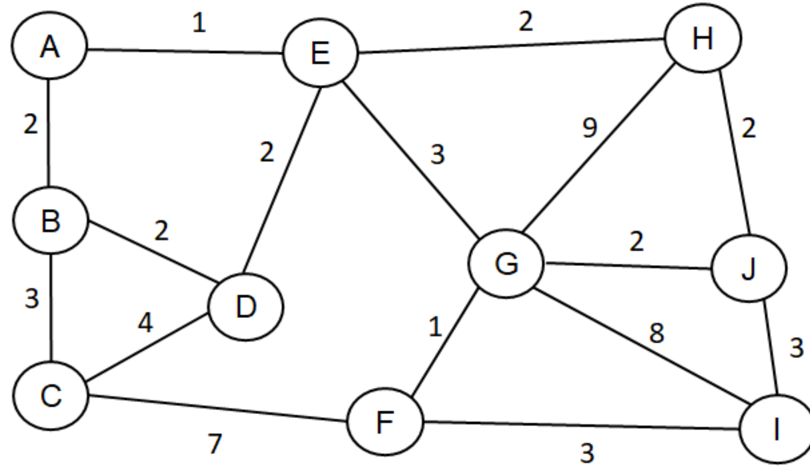
ii. Let w be the minimum weight edge among all the edge weights in G. If e is one of the edges of weight w, it will be present in every spanning tree of G.

(d) **(4 marks)** Consider the following graph. Which of the following is **not** the sequence of edges added to the minimum spanning tree on applying the Kruskal's algorithm.



i. (B,F) (E,F) (A,C) (B,C) (F,G) (C,D)
ii. (B,F) (E,F) (A,C) (F,G) (B,C) (C,D)
iii. (B,F) (A,C) (E,F) (B,C) (E,G) (C,D)
iv. (B,F) (E,F) (B,C) (A, C) (F,G) (C,D)

(e) **(4 marks)** Using Prim's algorithm find the cost of the minimum cost spanning tree of the following graph. You should show the edges and their weights (e.g., AC 6) in the order of being added into the tree by the algorithm considering A as the starting node.



* * * * * * * * * * * * * * *

Reference algorithms

1. BFS traversal for finding augmentation paths.

```
BFS(RG, s, t)
 augPath = ArrayList  of edges
 q := queue()
q.push(s)
 backpointer(v) = null for all  v    // backpointer data-structure to hold
                                     //edges that lead to the vertex
while ! q.isEmpty()
     cur := q.pull()
   for each outedge e of cur in RG do
     if  e.toCity! = s and backpointer(e.toCity) == null and e.cap ≠ 0
          backpointer(e.tocity):= e
          if( backpointer(t)! = null)   // found a path from s to t. Build it now from reverse
             pathEdge = backpointer(t)
             while( pathEdge ! = null)
                    augPath.add(pathEdge)
                    pathEdge = backpointer( pathEdge.fromCity)
             endWhile
              Collections.reverse(augPath)
             return augPath
          endIF
          q.push(e.toCity)
     endIf
   endFor
endWhile
return null
```

2. DFS traversal

```
DFSTraverseGraph(node):
   if  node is not visited:
      visit the node
      process the node
      for each neighbour of node:
         if  neighbour is not visited
            TraverseGraph(neighbour)
```

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.